Preliminary Design Review: Feature-Aided Tracking for the PCA Integrated Radar-Tracker Application

W. Coate
M. Arakawa

27 October 2004

Lincoln Laboratory

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LEXINGTON, MASSACHUSETTS

20041102 041

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER

Gary Tutungian
Administrative Contracting Officer
Plans and Programs Directorate
Contracted Support Management

Massachusetts Institute of Technology
Lincoln Laboratory

Preliminary Design Review:
Feature-Aided Tracking for the PCA Integrated Radar-Tracker Application

W.G. Coate
M. Arakawa
Group 102

27 October 2004

Lexington   Massachusetts

ABSTRACT

The Feature-Aided Tracker (FAT) system was developed at MIT Lincoln Laboratory as an addition to a simple kinematic tracker. The goal of FAT is to use high resolution information about a target's characteristics to aid both in the tracking and identification of targets.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

1. INTRODUCTION

The Feature-Aided Tracker (FAT) system was developed at MIT Lincoln Laboratory as an addition to a simple kinematic tracker [1]. The goal of FAT is to use high resolution information about a target's characteristics to aid both in the tracking and identification of targets.

An example of the type of problem FAT was designed to solve is illustrated in Figure 1. When two targets first converge toward a common point and then diverge after it, a kinematic tracker alone cannot reliably determine the path followed by each target.
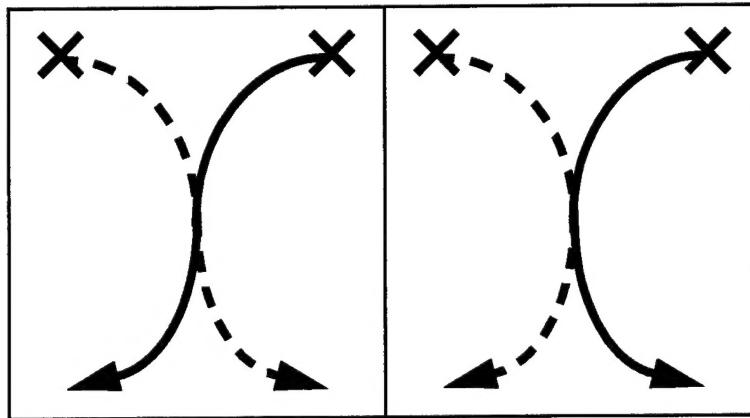
Figure 1. Kinematically ambiguous target motion example.

This report discusses the Feature-Aided Tracking component of the Integrated Radar-Tracker (IRT) application for the DARPA/IPTO PCA program. Only the portions specific to FAT will be discussed here. Descriptions of the portions of FAT common with a simple kinematic tracker can be found in [1]. An overview of the entire IRT application can be found in [3].

1.1 FEATURE-AIDED TRACKING—HIGH LEVEL DESCRIPTION

Feature-Aided Tracking uses one-dimensional High Range Resolution (HRR) profiles gathered for each target to improve the association likelihoods (referred to as χ^2 scores) generated by the kinematic tracker (see Figure 2). FAT has two components which may be used. Classification-Aided Tracking (CAT) compares incoming HRR profiles with profiles from a template class database. Signature-Aided Tracking (SAT) compares incoming HRR profiles with the HRR profile for the track's last associated measurement.

1

A weighted Mean Squared Error (MSE) metric is used for all HRR pattern matching operations. FAT may use either CAT, SAT, or both (using neither is essentially kinematic tracking).
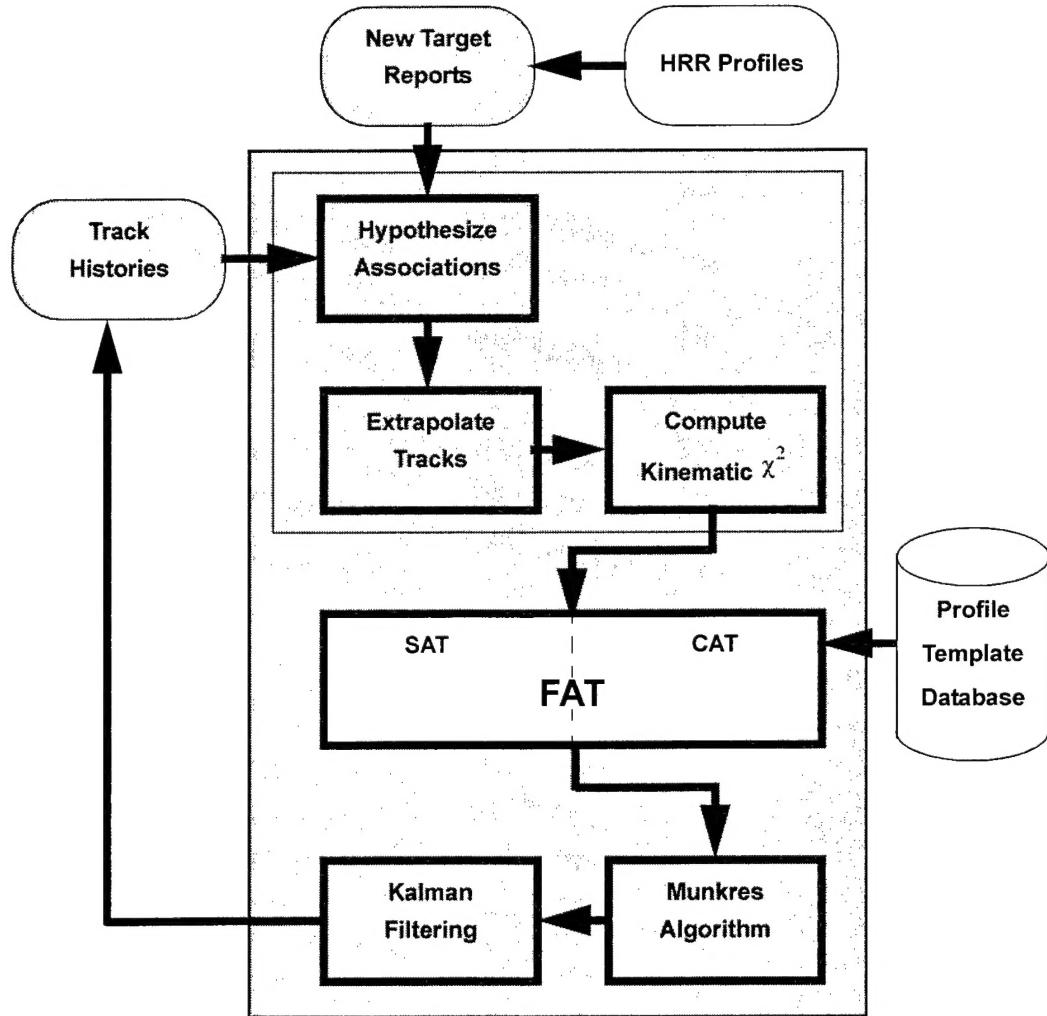
Figure 2. Feature-Aided Tracker logical flow diagram (FAT only components in white, kinematic tracker common components in gray).

1.1.1 Classification-Aided Tracking (CAT)

CAT is performed by forming a vector of MSE scores for the hypothesized target/track pair under consideration against all known classes within the template database. The minimum MSE score against templates within a given aspect angle range (nominally ±10°) of the target is used. A Bayesian classifier is then used on each MSE vector to determine the probability that each given hypothesized pair is one of the known target classes or an unknown class. This information is then used to compute an adjustment for the χ^2 score from the kinematic tracker.

CAT has the benefit of providing information about the type of object being tracked. Another positive point is providing greater accuracy and robustness for known target types. The requirement to gain these benefits from CAT is prior knowledge of target classes for the database. CAT may or may not be helpful in tracking for unknown target types (depending upon their characteristics and the characteristics of the known target classes) and cannot be used to identify unknown target classes. CAT also requires a significantly larger amount of processing than simple kinematic tracking or SAT.

1.1.2 Signature-Aided Tracking (SAT)

SAT is performed by taking the MSE comparison between the incoming target's HRR profile against the track's stored HRR profile for the hypothesized target/track pair. This information is then used to compute an adjustment for the χ^2 score from the kinematic tracker.

SAT has the benefit of requiring no prior knowledge of any target types. Also its processing requirements are relatively small compared to CAT. The largest disadvantage for SAT is that HRR profiles are not generally smooth across aspect angle[1]. As the aspect angle of an object changes, there will be points of discontinuity when new objects become part of the one-dimensional projection towards the radar. The result of this is less accuracy at unpredictable points. A system designer may attempt to bound the unpredictability by such methods as setting an aspect angle change threshold for the use of SAT, but no solution can be perfect.

1.2 PARAMETERS

This section explains parameters which are needed as inputs and controls for the operation of FAT. As shown in Figure 2, a tracker works as a loop-back system, where the output for one scan may be taken as the input for the next scan, therefore the output will also be discussed. The input and output parameters are a superset of those required for the kinematic tracker. Some control parameters are common to the two systems as well.

---

[1] To simplify the code and allow constant SAT processing, the patterns in the sample data are smooth across aspect angle.

1.2.1   Input Parameters

FAT requires two input structures, a target report list and the track list output by the previous scan. The target report list will be signified by the one-dimensional array M[], and indices into M[] will be generally signified by "M_Idx" or M_Idx. The parameters shown in Table 1 are the inputs to the kinematic tracker from an external system (test data system or a radar). Shaded rows of Table 1 are inputs used primarily by the kinematic tracker (so their results are used by FAT, but they are not directly used). The track list will be described in Section 1.2.2.

TABLE 1

Data Input To Tracker (rows in gray for primarily kinematic tracker input)

Parameter name	Explanation
rg	Range position of target. The kinematic tracker uses this to determine ground position.
az	Azimuth position of target. The kinematic tracker uses this to determine ground position.
dop	Radial velocity of target. The kinematic tracker uses this to estimate target velocity vector.
time	Time stamp for when this detection was made.
snr	Signal-to-noise ratio (SNR) of detection. This is a measure of the relative strength of the target detection.
hrr	High Range Resolution profile for target. This is the "pattern" FAT uses for the incoming target for all pattern matching operations.

1.2.2   Output Parameters

The output is also in the form of an array of structures called the track list. It will generally be referred to as the one-dimensional array T[], and indices into T[] will generally be signified by "T_Idx" or T_Idx. The purpose of the track list is two-fold. One is as the output to a user; the other is as loop-back input to the tracker. Table 2 outlines data members of T[] which are relevant outside the current scan of the tracker (either to user or to next tracker scan). Shaded rows of Table 2 are not directly related to FAT, but to the kinematic tracker.

4

TABLE 2

Data Output from Tracker (rows in gray are of only indirect interest to FAT)

Parameter name	Explanation
snr	SNR of last target report associated with this track. This is a measure of the relative strength of that target detection.
Pp	Extrapolated process noise covariance matrix. This is used only by the kinematic tracker (primarily by the Kalman filter).
status	{New, Novice, Established}
time	Time stamp for the last target report associated with this track.
x	Estimated x-coordinate of target at time time.
y	Estimated y-coordinate of target at time time.
x_dot	Estimated velocity of target in the x direction at time time.
y_dot	Estimated velocity of target in the y direction at time time.
Hypothesis	Kinematic movement model hypothesized for last target/ track association for this track. Used to determine degrees of freedom for merging χ^2 scores.
classificationVector	Classification vector from CAT. A vector of probabilities that the track belongs to each known class as well as to an unknown class.
hrr	High Range Resolution profile for last target report associated with this track. This is the "pattern" FAT uses for the track when performing SAT.

1.2.3   Control Parameters

FAT uses many control parameters. In the implementation, these are part of a structure (see Section 3.4). For the purpose of algorithmic descriptions, they will be treated as separate values. Parameters important to the algorithmic description or general conceptual understanding are listed in Table 3.

5

TABLE 3
Control Parameters for the Feature Aided Tracker

Parameter name	Explanation
MaxNumReports	Maximum number of target reports allowed (necessary only for real-time bound or worst case analysis of memory restrictions of actual implementations). This parameter is not enforced in the Matlab implementation.
MaxNumTracks	Maximum number of tracks allowed (necessary only for real-time bound or worst case analysis of memory restrictions of actual implementations). This parameter is not enforced in the Matlab implementation.
UsingCAT	Turn on or off Classification-Aided Tracking.
UsingSAT	Turn on or off Signature-Aided Tracking.
NumClasses	Number of classes known by CAT template database.
NumOtherClasses	Number of classes generated for use in creating "other" target statistics for CAT.
HRRPixels	Number of samples in HRR profile.
MatchMu	Mean MSE score for matching HRR profiles in SAT.
MismatchMu	Mean MSE score for HRR profiles not matching in SAT.
MatchVar	Variance of MSE scores for matching HRR profiles in SAT.
MismatchVar	Variance of MSE scores for HRR profiles not matching in SAT.
PriorClamp	Value used in place of zero for prior probabilities (zero causes failure to ever consider possibility again).
AspectScan	Assumed potential aspect angle estimation error for use in choosing template profiles from CAT database.
ShiftRatio	Indicates degree to which HRR profiles may be shifted in range to find the minimum MSE score.
MagDBLow	Lowest magnitude power scaling to find the minimum MSE score.
MagDBHigh	Highest magnitude power scaling to find the minimum MSE score.
MagDBStep	Step size for determining values between *MagDBLow* and *MagDBHigh* to examine.
default_prior	Size (*NumClasses* + 1) vector of probabilities to use as default for new tracks.

2. FUNCTIONAL OVERVIEW

This section will cover the basic functionality required specifically for FAT. Figure 3 illustrates FAT's internals in slightly greater detail than shown in Figure 2. Note that this picture differs from Figure 2 in that the Munkres Algorithm is shown here where it is actually performed, within FAT. This implementation eases the updating of some FAT-specific data elements. Figure 2 is a conceptual overview of the manner in which FAT functions, where Figure 3 reflects more precisely the details of the provided implementation.
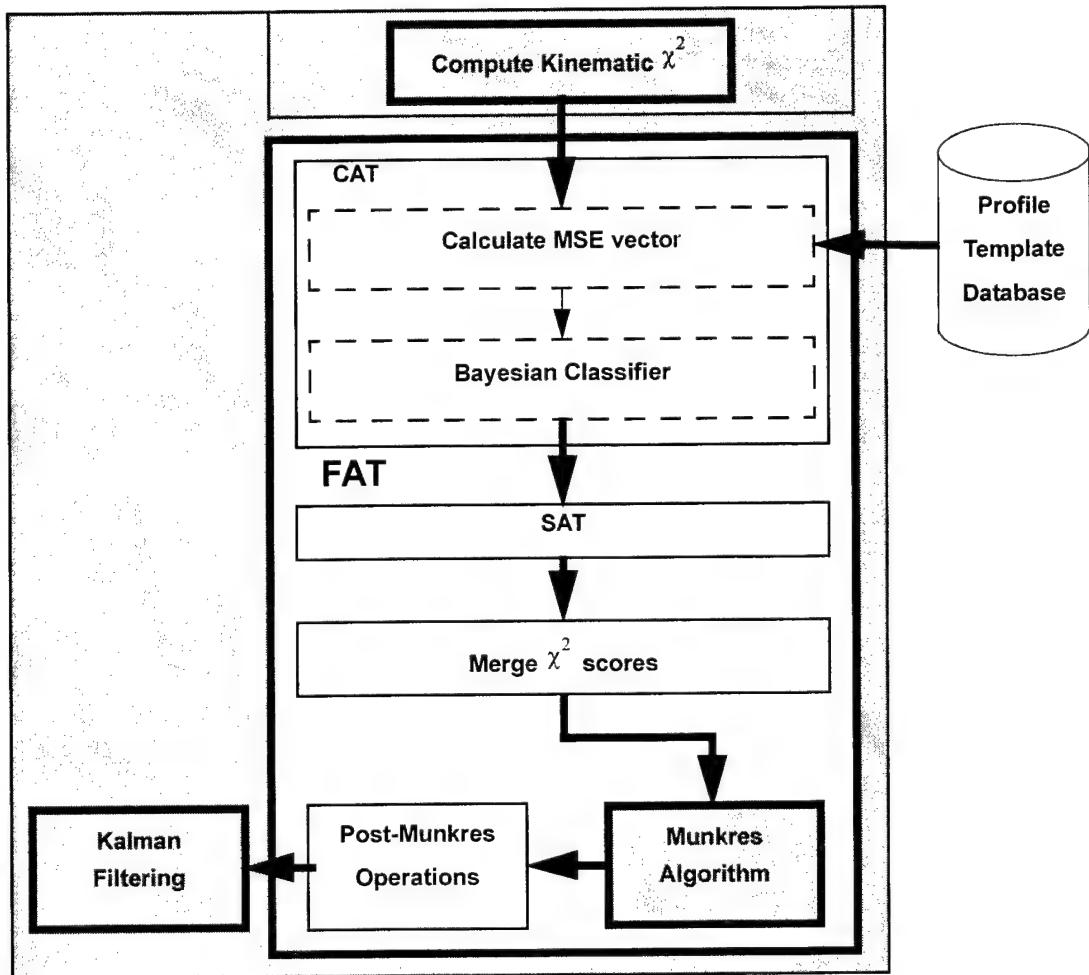
Figure 3. FAT logical block diagram (kinematic tracker portions in gray).

Functionality shared with the kinematic tracker (including the Munkres Algorithm) has been described in the kinematic tracker's documentation [1]. Implementation choices will be discussed in Section 4.

2.1 COMPUTE MSE SCORE [`calculateMSE()`]

Input: An HRR profile under test (size 1x*HRRPixels*) and a database of an arbitrary number of HRR templates (each size 1x*HRRPixels*).

Output: A floating point number indicating the minimum MSE score for the test profile against all template profiles.

All pattern matching for CAT and SAT use a weighted MSE metric as specified below.

$$\text{weighted MSE} = \frac{\displaystyle\sum_{k=1}^{HRRPixels} w_k(t_k - h_k)^2}{\displaystyle\sum_{k=1}^{HRRPixels} w_k} \tag{1}$$

where

- w_k is the weight for the kth pixel
- t_k is the magnitude of the kth pixel in the track history HRR profile
- h_k is the magnitude of the kth pixel in the target HRR profile

Each weight w_k is computed as

$$w_k = 1 - pr_k \times pt_k \tag{2}$$

where

$$pr_k = \begin{cases} 1 & t_k \leq 2\eta \\ 0 & \text{otherwise} \end{cases}, \tag{3}$$

$$pt_k = \frac{1}{8} e^{-2t_k/\eta}, \tag{4}$$

and η is the clutter power level.

8

An HRR profile collected by a sensor may be translated in range or power compared to the stored template sample. The optimal values for [range] shift and [power] gain are found through brute force comparisons, returning the minimum MSE score. Due to the fact that the MSE metric forms a parabolic function of shift and gain, optimum values may be found by shifting first the range and then finding the optimal power gain for the resulting minimum (see Figure 4).
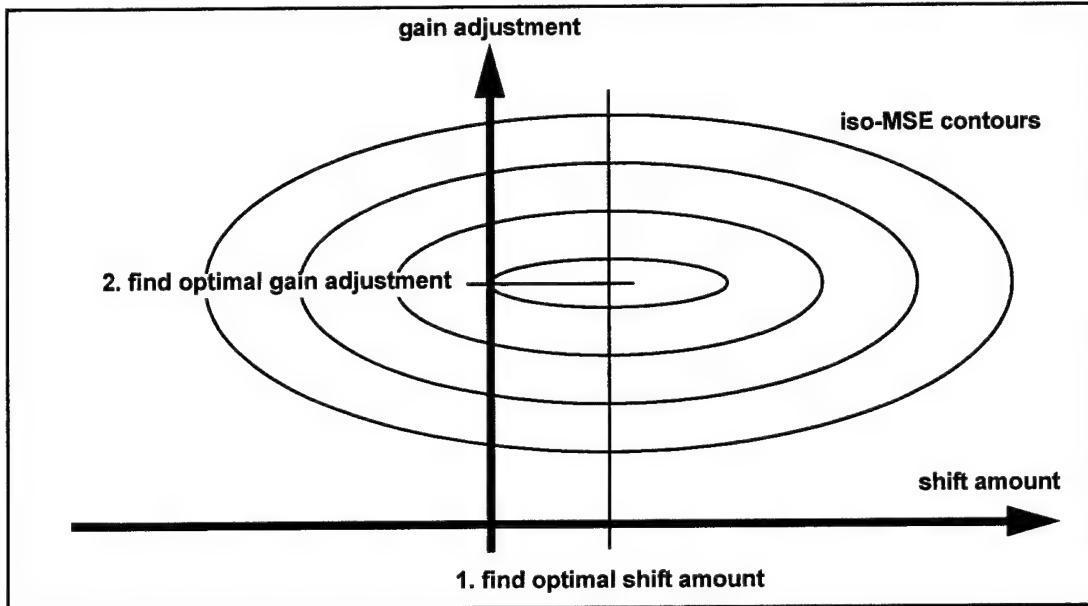
Figure 4. Two-step method for finding minimum MSE score.

This explanation is for performing a single MSE comparison between two HRR profiles. Performing multiple comparisons is essentially done by performing this multiple times, though some results may be re-used depending upon implementation (shifted profiles for testing are re-used for the supplied Matlab version).

2.2   CLASSIFICATION-AIDED TRACKING (CAT)

Input: A target report's HRR profile (size 1x*HRRPixels*), the prior probabilities (size 1x*NumClasses*+1) for the track hypothesizing association, and the estimated aspect angle for the association.

Output: A χ^2_{CAT} value for the hypothesized track/target pair and a set of posterior probabilities (size 1x*NumClasses*+1) for the hypothesized association.

9

Classification-Aided Tracking performs a comparison between an incoming target report and the profiles stored in a database. The comparison must be performed for each track/target pair, as opposed to only for each target, because the method for estimating aspect angle requires an associated track[2]. CAT calculates a vector of MSE scores (see Section 2.2.1) which are then used by a Bayesian classifier to attempt to clarify target classification (see Section 2.2.2). Using the Probability Density Function (pdf) output from the Bayesian classifier and the track's prior probabilities vector, $\chi^2{}_{CAT}$ is computed as follows:

$$\chi^2{}_{CAT}(\text{T\_Idx, M\_Idx}) = -\ln (\text{pdf} \cdot \text{prior}) \tag{5}$$

where "$A \cdot B$" indicates the vector inner product of A and B.

For any parameter set chosen, CAT will require the largest share of the tracker's processing. For nontrivial parameter sets, CAT will dominate the processing by a large margin (easily over 95% of the total operation count). Which component dominates depends upon the parameter set. For all parameter sets we have investigated, MSE vector calculation dominates (more than three quarters the total operation count for FAT). However, if the number of known target classes is increased while other parameters are left alone, the Bayesian classifier will dominate. This is because MSE vector calculation is linearly dependent upon three major parameters (number of known classes, number of hypothesized pairs examined, and number of distinct aspect templates to examine) while the Bayesian classifier's major dependence is on the number of classes to the fourth power. In a real-world system, for the Bayesian classifier to dominate computationally, this would imply that FAT is using a database with a large number of targets but poor aspect resolution to examine a small number of widely spread targets. For most cases of interest, it is safe to consider MSE vector computation FAT's computational driver.

2.2.1 Calculate MSE Vector [`computeMSEvector()`]

Input: A target report's HRR profile (size 1x*HRRPixels*) and the estimated aspect angle for the association.

Output: A vector of MSE scores (size 1x*NumClasses*) for the hypothesized association's score against known target classes (named `MSEVec` in the pseudocode below).

With the parameters and scenarios provided, this function is responsible for more than three quarters of FAT's processing. The reason is that for each scan, an MSE score computation (as described in Section 2.1) must be computed for each associated pair against each template class at each possible aspect angle stored in the database. For each hypothesized pair and each template class, the minimum score within the considered aspect angle range will be returned.

---

[2] Aspect angle is assumed to match velocity vector direction. This estimate is obtained by running the hypothesized pair through a Kalman filter [1] and using the resulting velocity vector.

A Matlab pseudocode outlining the logical calculation of all MSE vectors is given below.

```
for each track T_Idx
    for each possibly associated target report M_Idx
        Aspect= estimated aspect for pair (T_Idx,M_Idx);
        %BEGIN MSEVector CALCULATION HERE
        for each template class C_Idx
            S= arbitrary large number
            for each profile within AspectScan of Aspect A_Idx
                    R=calculateMSE(M(M_Idx).hrr, ...
                        Template(C_Idx,A_Idx));
                    S=min(S,R);
            end
            MSEVec(C_Idx)=S;
        end
        %END MSEVector CALCULATION HERE
        All_MSEVec{T_Idx,M_Idx}=MSEVec;
    end
end
```

The provided implementation is discussed in Section 4.

This function is also the primary point of convergence between the tracker portion of the IRT and the kernels described for the PCA program in Project Report PCA-KERNEL-1 [4]. FAT's `calculateMSE()` function was simplified to formulate the pattern matching kernel. Given the massive number of MSE scores to calculate, an efficient pattern database optimized to retrieve aspect angle ranges for given classes could provide great benefit.

2.2.2 Bayesian Classifier [computeBayes()]

Input: A vector of MSE scores (size 1x*NumClasses*) for the hypothesized association's score against known target classes, the prior probabilities (size 1x*NumClasses*+1) for the track, and the estimated aspect angle for the association.

Output: A probability density function (pdf) vector (size 1x*NumClasses*+1) and a posterior probability vector (size 1x*NumClasses*+1) describing the hypothesized association's chances of being one of the known classes or a completely unknown class.

Bayes' theorem for an exhaustive list of mutually exclusive events (such as a target being either one of *NumClasses* distinct classes or none of them, represented by A_i below) may be stated generally as follows:

11

$$P(A_i|B) = \frac{P(A_i)P(B|A_i)}{\sum_{i=1}^{N} P(A_i)P(B|A_i)} \tag{6}$$

where P(X) indicates the probability of X and P(Y|Z) indicates the probability of Y if Z is given/observed [2].

Substituting in specific values for FAT would give the following:

$$P(Class_i|MSEVec) = \frac{P(Class_i)P(MSEVec|Class_i)}{\sum_{i=1}^{NumClasses+1} P(Class_i)P(MSEVec|Class_i)} \tag{7}$$

where $Class_{NumClasses+1}$ indicates the "other" class, signifying any target class which is not represented by a template in the database.

The track's stored prior probabilities[3] indicate the probability of the hypothesized association being a given class (or $P(Class_i)$, $\forall i$). The probability that a given class would yield the MSE vector input (or $P(MSEVec|Class_i)$) is obtained from the following function:

$$P(MSEVec|Class_i) = \frac{e^{-\frac{1}{2} \cdot \Delta x_i \cdot Cov_i^{-1} \cdot \Delta x_i^{T}}}{\sqrt{(2\pi)^{NumClasses} \det(Cov_i)}} \quad {}^{4} \tag{8}$$

where Δx_i is the difference between the given MSE vector and mean MSE vector for $Class_i$, Cov_i is the stored covariance matrix for MSE vectors observed from $Class_i$, det(A) indicates the determinant of matrix A, and $Class_{NumClasses+1}$ indicates the "other" class.

The vectors given by collecting all values of $P(MSEVec|Class_i)$ and $P(Class_i|MSEVec)$ are returned as the pdf and posterior probabilities, respectively.

---

[3] Prior probabilities indicate the probability of an event given no additional knowledge. In the case of FAT, a track's prior probabilities indicate the probability that the track indicates a target of each known class and the "other" (or unknown) class without taking into account HRR information from the current scan. Posterior probability is the probability estimate after taking the current scan's information into account.
[4] A slightly different implementation was used for numerical stability. This will be discussed in Section 4.

2.3   SIGNATURE-AIDED TRACKING (SAT)

Input: The HRR profile (size 1x*HRRPixels*) associated with the track and the HRR profile (size 1x*HRRPixels*) associated with the target report for the hypothesized association pair.

Output: A $\chi^2{}_{SAT}$ value for the hypothesized track-target pair.

Signature-Aided Tracking performs a direct comparison between an incoming target and a saved track. Depending upon the HRR profiles' expected smoothness across aspect angle, a limit may be imposed on what pairs SAT operates upon, though this is omitted in the provided implementation due to the use of HRR profiles changing smoothly over aspect. The believed statistical likelihoods of receiving the returned MSE score are used to generate the probability density of a match or mismatch. The probability densities are then used to calculate the $\chi^2{}_{SAT}$ adjustment as shown below.

$$\chi^2{}_{SAT}(\text{T\_Idx, M\_Idx}) = -\ln (\text{Match\_pdf} / \text{Mismatch\_pdf}) \tag{9}$$

where

$$\text{Match\_pdf} = \frac{e^{-\frac{1}{2} \cdot \frac{(\text{hrr\_mse} - MatchMu)^2}{MatchVar}}}{\sqrt{2\pi MatchVar}} , \tag{10}$$

$$\text{Mismatch\_pdf} = \frac{e^{-\frac{1}{2} \cdot \frac{(\text{hrr\_mse} - MismatchMu)^2}{MismatchVar}}}{\sqrt{2\pi MismatchVar}} , \tag{11}$$

and

$$\text{hrr\_mse} = \log (\text{calculateMSE}(M(\text{M\_Idx}).\text{hrr\_profile}, T(\text{T\_Idx}).\text{hrr\_profile})) \tag{12}$$

for hypothesized track/target pair (T(T_Idx),M(M_Idx)) and using the MSE calculation outlined in Section 2.1.

2.4   MERGE χ^2 SCORES

Merging the χ^2 scores from the kinematic tracker with $\chi^2{}_{SAT}$ and $\chi^2{}_{CAT}$ is performed through a simple addition. In a real system, other adjustments of the various χ^2 scores might be placed here as well. Currently some adjustments are made to χ^2 scores based upon what hypothesis the kinematic tracker

chose. It may be desirable to scale the three χ^2 scores to a similar order of magnitude, and that scaling would be performed at this stage in the code.

2.5   POST-MUNKRES OPERATIONS [`compute_track_diffusion()`]

Input: Saved posterior probability vectors for all track/target pairs, final χ^2_{FAT} matrix.

Output: Updated prior probabilities for each track on the next scan.

All FAT-specific data elements, save the track's prior probabilities, are updated through simple assignments. The updated prior probabilities for the track provide not only input for the next scan, but also output information for the overall system. Because it should be considered possible that the right association was not made (perhaps due to a target not being detected in the current scan), the updating of priors for the next scan should take into account all possibilities and not only the final chosen association. This is done by comparing the magnitudes of the potentially associated χ^2 scores in the following manner:

$$T.\text{Priors} = \sum_{\forall i \, \in \, \text{PotentialAssoc(T\_Idx)}} \frac{1/\chi^2_{FAT}(\text{T\_Idx}, i)}{\Sigma\chi^2} \cdot T.\text{PosteriorSaved}(i) \tag{13}$$

where

$$\Sigma\chi^2 = \sum_{\forall i \, \in \, \text{PotentialAssoc(T\_Idx)}} 1/\chi^2_{FAT}(\text{T\_Idx}, i). \tag{14}$$

14

3. DATA STRUCTURES

A variety of data structures are used in FAT, many of them shared with the kinematic tracker. Below is a description of their actual structure. These definitions will be given in a C style to clearly indicate data types and sizes. (Note that, in C, pointers are generally used to declare dynamically resizable arrays.)

3.1 TARGET REPORTS

The structure for target reports is simple and mostly used by the kinematic tracker as opposed to FAT (only the HRR profile is used by FAT alone). The global list of target reports has been referred to as M[] in this document, and indices for specific locations have been given the name M_Idx (indices solely for looping may use other names). The centroid structure contains the information which is actually passed from a radar system to the tracker and will be saved as a data member of elements of M[], which stores additional information derived from the centroid structure's information (such as cosine of the azimuth angle).

```
struct target_report
{
  struct centroid cent;    //contains original info passed
                           // in from MTI
  double Range;            //These three parameters are the
  double Azimuth;          // ground values of rg, az, & dop
  double Doppler;          // in cent (the values in cent
                           // may be absolute or may be
                           // indices).
  double sin_az;           //sine of the azimuth angle
  double cos_az;           //cosine of the azimuth angle
  double sdsqd;            //doppler variance (in m/s)
  double abs_dop;          //corrected doppler (in m/s)
  double R[2][2];          //measurement (x,y) covariance
                           // matrix

}
```

```
struct centroid
{
  int rg;                      //range gate index or absolute
                              // range in meters for this
                              // target
      int az;                  //clutter-nulled beam index or
                              // absolute azimuth angle in
                              // radians for this target
      int dop;                 //doppler bin index or absolute
                              // doppler in m/s for this target
      double time;             //time stamp associated with
                              // this target
      double snr;              //SNR for this target
      double hrr[HRRPixels];   //vector containing the HRR
                              // profile for this target
    //additional members to be populated by the tracker
    // all this information should be redundant with the
    // above info.
      double x;                //x coordinate of this target
      double y;                //y coordinate of this target
}
```

3.2   TRACK

The track structure is a simple structure, the majority of which is used for only the kinematic tracker. The global list of which has been referred to in this document as T [] and indices for specific locations have been given the name T_Idx (indices solely for looping may use other names). This structure is used both as output and input, though only a subset of the structure is required in the following scan.

```c
struct track_history
{
 char status[];                     //string containing status
                                    // (enum should be used in
                                    // compiled implementation)
 double x_pos;                      //x coordinate of estimated
                                    // position of track
 double y_pos;                      //y coordinate of estimated
                                    // position of track
 double x_vel;                      //estimated x direction
                                    // velocity of track
 double y_vel;                      //estimated y direction
                                    // velocity of track
 double snr;                        //snr from last associated
                                    // target
 double time;                       //time from last target
 struct VelWindow vw[5];            //represents different
                                    // possible actual doppler
                                    // values, used in resolv-
                                    // -ing doppler ambiguity
 struct CVHFilter cvhf;             //Abstraction from original
                                    // system, may be removed.
 struct target_report Msave[5];
                                    //saves old targets for
                                    // New/Novice tracks until
                                    // they become established
 char Hypothesis[];                 //string containing
                                    // hypothesis (enum should
                                    // be used in compiled
                                    // implementation)
 double Q[4][4];                    //process noise covariance
                                    // matrix
 double Pm[4][4];                   //initial extrapolation
                                    // covariance matrix
 double Pp[4][4];                   //updated extrapolation
                                    // covariance matrix
 double K1[4][2];                   //Kalman gain stage 1
                                    // matrix
 double K2[4][1];                   //Kalman gain stage 2
                                    // matrix
```

```c
    // New for FAT
    double hrr[HRRPixels];            //HRR profile from last
                                      // associated target
    double priors[NumClasses+1];      //Prior probabilities for
                                      // each class & 'other' for
                                      // the track
    double **posteriorSave;           //Location to save the
                                      // posterior vectors for
                                      // each possible target
                                      // association
    double *CAT_X2_Save;              //saves CAT_X2 for each
                                      // possible association
    double classificationVector[NumClasses+1]
                                      //Output to user which
                                      // indicates current belief
                                      // of track's class
    double cum_assoc;                 //sum of track's CAT_X2
    int TMAindex;                     //index of last associated
                                      // target report
    struct target_report assoc;       //copy of last associated
                                      // target report
    double Chisqd;                    //X^2 value for last
                                      // association
    double SAT_X2_Save;               //SAT_X2 value for last
                                      // association
}
struct VelWindow
{
 int active;                         //indicates state of this
                                     // velocity window
 double xdot;                        //x direction velocity
 double ydot;                        //y direction velocity
 double xdotvar;                     //variance in x direction
                                     // velocity
 double ydotvar;                     //variance in y direction
                                     // velocity
 double xydotvar;                    //covariance between x and
                                     // y velocities

}
```

```
struct CVHFilter
{
 double x;                              //x coordinate
 double y;                              //y coordinate
 double xdot;                           //x velocity
 double ydot;                           //y velocity
}
```

3.3 ASSOCIATION MATRICES

These matrices are simply $t \times m$ matrices, where t is the length of the global track list (T [ ]) for the current scan and m is the length of the global target report list (M [ ]) for the current scan. Because the sizes of these lists may change from scan to scan, the dimensions of these matrices also may change. There will be seven matrices of this size. For simple kinematic tracking, the matrices are Assoc, Hyp, and Kin_X2. For FAT, the additional matrices CAT_X2, SAT_X2, DegFreedom, and FAT_X2 are also used. A dynamically resizable matrix in C is represented by a double pointer. Due to the fact that a string in C is usually represented by a char*, a matrix of strings will be represented as a triple pointer. A matrix of arrays would also be represented as a triple pointer.

```
bool **Assoc;       //T/F matrix based on geographic and
                    // kinematic feasibility
char* **Hyp;        //matrix of strings keeping track of
                    // which hypothesis was last chosen
                    // (enum should be used in compiled
                    // implementation)
double **Kin_X2;    //contains X^2 values for kinematic
                    // tracker
double **CAT_X2;    //contains X^2 adjustments from CAT
double **SAT_X2;    //contains X^2 adjustments from SAT
int **DegFreedom;   //used to normalize Kin_X2 scores
                    // according to Hyp and FAT adjustments
double **FAT_X2;    //contains final/merged X^2 values
```

3.4 FAT PARAMETERS

This simple structure stores control parameters set during initialization. Kinematic tracker parameters needed by FAT are duplicated within this structure. None of these parameters are expected to change after initialization.

```
struct FATParameters
{
    // basic/common FAT parameters
bool UsingCAT;                    //Turn CAT on/off
bool UsingSAT;                    //Turn SAT on/off
int HRRPixels;                    //HRR vector size
    // parameters for calculateMSE()
double ShiftRatio;               //Relative amount to poten-
                                 // tially shift HRR profile
double MagDBLow;                 //Low power scaling
double MagDBStep;                //Power scaling delta
double MagDBHigh;                //High power scaling
    // Statistics used by SAT
double MatchMu;                  //Mean of log10 of MSE score
                                 // from HRR profiles for
                                 // matching targets
double MismatchMu;               //Mean of log10 of MSE score
                                 // from HRR profiles for
                                 // differing targets
double MatchVar;                 //Variance of log10 of MSE
                                 // score from HRR profiles
                                 // for matching targets
double MismatchVar;              //Variance of log10 of MSE
                                 // score from HRR profiles
                                 // for differing targets

    // CAT control parameters
int NumClasses;                  //Number of known classes
bool TrackDiffusion;             //Turn on or off updating of
                                 // prior probabilities
double PriorClamp;               //Minimum probability
                                 // allowed in a prior
int NumSectors;                  //Number of sectors used to
                                 // consider areas of object
int AspectScan;                  //Degrees off aspect
                                 // estimate to examine for a
                                 // match
```

```cpp
double default_prior[NumClasses+1];
                                    //Default prior probability
    // Parameters used for degree of freedom calculation
    //    in X^2 fusion
double DF_Tag;                      //Used for pairs with zero
                                    // degrees of freedom
int NumChiSqd;                      //Number of values in X^2
                                    // distribution
    // legacy parameters from kinematic tracker
double PlatformXPos;               //X position of radar by the
                                    // XY grid used with targets
double InvalidChiSqd;              //X^2 value to indicate an
                                    // impossible pairing
    // Parameters for the profile generator used for
    //    perfect data
double HRRGenProfileHi;            //High/peak value
double HRRGenProfileLo;            //Low/valley value
double HRRGenNoiseLevel;           //Radar noise level
    // Profile database generator parameters
bool CreateDB;                      //Turn on/off database
                                    // generator
int NumOtherClasses;               //Number of classes past
                                    // NumClasses to use for
                                    // creating statistics for
                                    // unknown classes
double AspectEvenDist;             //Distance used between
                                    // aspect measurements for
                                    // evenly distributed
                                    // template databases
double AspectMinDist;              //Minimum distance used
                                    // between aspect measure-
                                    // ments for unevenly
                                    // distributed template
                                    // databases
double AspectMaxDist;              //Maximum distance used
                                    // between aspect measure-
                                    // ments for unevenly
                                    // distributed template
                                    // databases
double CovEigDynRgLimit;           //Limit used for covariance
                                    // matrix diagonal loading
```

```
double MinCovDet;                  //Desired minimum for co-
                                   // variance matrix
                                   // determinant
double MinDiagLoad;                //Minimum allowable value
                                   // used for covariance
                                   // matrix diagonal loading
int FileAngleExtent;               //Granularity for angle
                                   // separation into files
char* TemplateDirectory;           //Directory for template
                                   // database
char* StatsDirectory;              //Directory for template
                                   // statistics database
                                   // & SAT statistics
double StatSampleMultiplier;       //Multiplier by number of
                                   // classes for number of
                                   // samples to use for
                                   // statistic creation
    // Parameters relating to parallelization with
    // MatlabMPI (see http://www.ll.mit.edu/MatlabMPI/
    // for more information on MatlabMPI)
struct MatMPIComm comm;            //Structure required by
                                   // MatlabMPI (only used to
                                   // pass to MatlabMPI func-
                                   // tions)
int NumProc;                       //Number of processes
int MyRank;                        //Rank of this node
int MPICmd;                        //MPI Tag used to
                                   // indicate a command
int MSE_Tag;                       //MPI Tag used to
                                   // indicate MSE
                                   // operation to be
                                   // performed
    //NOTE:  My usage of MatlabMPI tags is not the
    //       recommended manner, so should be used
    //       carefully (if at all) as a MatlabMPI example
```

4. IMPLEMENTATION CONSIDERATIONS

In this section we discuss in more detail three features of the Matlab implementation and relate them to implementations on actual PCA hardware. The first consideration brought up is a combination of clarity and efficiency for joining FAT with the kinematic tracker. The majority of the section focusses on the area around MSE vector computation, due to the fact that this is where the majority[5] of computation time is spent. The last paragraph discusses reduction in the computation time required for the Bayesian classifier.

The logical view of FAT (shown in Figure 2) is appropriate to understanding how the overall tracking system operates. However, in terms of actual functional implementation, it is simpler to call the Munkres Algorithm from within FAT as opposed to immediately after it (as shown in Figure 3). It would be possible to separate the current FAT function into two separate FAT functions which would be called before and after the Munkres Algorithm, or to incorporate these portions of FAT into the kinematic track updates along with the Kalman filter. In an attempt to keep simplicity of high-level function layout and to keep the FAT data updating separate from the functions used for the kinematic tracker, the Munkres Algorithm is performed within FAT (or instead of FAT when running only the kinematic tracker). Different choices may be desirable for efficiency on specific hardware implementations.

The structure of FAT was changed to facilitate ease of parallelization choice within the `computeMSEvector()` function. The logical view shown in the pseudocode in Section 2.2.1 is the way it was originally structured. The sample implementation restructured the higher level functions to place all loops within the boundaries of `computeMSEvector()` and then used the number of template classes as the parallelization axis as shown in the notional changed pseudocode below.

---

[5] Matlab profiler results with no optimizations have consistently shown MSE vector computation to take a minimum of 95% of the processing time, and spreadsheet analysis puts it over 75% for the parameters and scenarios examined.

```
for each track T_Idx
    for each possibly associated target report M_Idx
        Aspect= estimated aspect for pair (T_Idx,M_Idx);
        Save pertinent info to struct
    end
end
%BEGIN MSEVector CALCULATION HERE
while template classes C_Idx left to process
    for each parallel node
        Send data for one template class processing
    end
    for each Pair (T_Idx,M_Idx) in input struct
        S= arbitrary large number
        for each profile within AspectScan of Aspect A_Idx
            R=calculateMSE(M.hrr,Template(C_Idx,A_Idx));
            S=min(S,R);
        end
        MSEVec(T_Idx,M_Idx)=S;
    end
    All_MSEVec{:,:}(C_Idx)=MSEVec';
    for each parallel node
        Receive MSEVec results for processed class
        All_MSEVec{:,:}(C_Idx+node_offset)=MSEVec';
    end
end
%END MSEVector CALCULATION HERE
```

The reason for performing the computation in this manner is that in a real system each parallel node could possess a local copy of the class(es) it deals with, thus reducing communication. This restructuring, by placing all loops within the same function, should also ease the changing of parallelization axis among implementations. This parallelization axis flexibility could allow a run-time system to choose different axes depending upon the resources available. An intelligent fine-grain control system could take advantage of this loop structure to eliminate duplicate database queries and balance global communication needs. Such a system could allow a minimum of communication by maximizing the MSE computations made before new profiles are loaded and minimizing computation by reusing MSE scores as much as possible for pairs with the same detection report. The sample implementation was parallelized, as in the pseudocode shown above, but contains none of these advanced features. The revised loop structure should facilitate any axis of parallelization desired, or the use of multiple axes. If one parallelization axis is obviously the best choice for a given implementation, it may be desirable to readjust loops within the greater FAT structure and suitably adjust the boundaries of the computeMSEvector() function.

To preserve clarity and keep all functionality transparent at a Matlab level, the template profiles "database" is simply the file system. Each class was separated into files containing ten degrees of aspect angle results. The value of ten degrees is relatively arbitrary. This system was in no way tuned for performance. We attempted to keep the system open, flexible, and transparent. However, for performance-oriented demonstrations, some efficient manner of data storage, or at least data accounting, will be needed.

On a traditional system, the implementation would require either a good database method for pattern retrieval or would parallelize so each node has all needed template profiles in fast, local storage. A PCA system could allow a more intelligent orchestration of all data and computation. An "orchestrator" process may parcel out specific MSE computations to be performed on given nodes, indicate when templates need to be sent to specific resources, and organize the output to minimize both communication and computation. The ability to use PCA's flexibility and "morphing know how" to schedule and synchronize communication and computation for dynamic problems would be a very impressive display of PCA chip abilities.

A performance choice made solely on the Matlab side was to use Matlab's MEX file optimization for the `calculateMSE()` function. MEX-files in Matlab are an interface for C code to be compiled to function as a native Matlab function. It should be noted that the loop-over templates within *AspectScan* of the estimated aspect angle and taking the minimum is actually performed within the `calculateMSE()` function and was only listed separately here for clarity of function. We have observed FAT speedups roughly on the order of 20x by using this MEX file. The Matlab M-file version was renamed to `old_calculateMSE.m` and is included as part of the sample implementation.

The computation in the Bayesian classifier has the potential to scale to dominate FAT if the number of known target classes in the database is increased significantly past the number of hypothesized associations and templates by aspect angle. There are multiple simple solutions which could be employed to reduce this computation time. First, we explain the actual operation performed. Using Equation 8, problems were encountered because the determinant of some of the stored covariance matrices were falling below Matlab's minimum real number threshold and being reported as zero. The following function was used to avoid the determinant:

$$P(MSEVec|Class_i) = \frac{e^{-\frac{1}{2} \cdot \left(\sum \log(\text{eigenvalues}(Cov_i)) + (\Delta x_i \cdot Cov_i^{-1} \cdot \Delta x_i^{\mathrm{T}}) \right)}}{\sqrt{(2\pi)^{NumClasses}}}. \tag{15}$$

Equation 15 contains two operations which run cubic in the number of template classes, eigenvalue computation and matrix inversion. When performed for each class, this gives a dependence on the number of classes raised to the fourth power. These could be eliminated through more offline computation and stored data, lowering the highest order term in Equation 15 to square, resulting in a net cubic dependance upon the number of classes. Other options for likelihood estimation could be relatively easily examined for goals of computation reduction and numerical stability.

ACRONYMS

CAT – Classification-Aided Tracking

DARPA/IPTO – Defense Advanced Research Projects Agency/Information Processing Technology Office

FAT – Feature-Aided Track[er/ing]

HRR – High Range Resolution

IRT – Integrated Radar-Tracker

MEX – Matlab EXtension

MSE – Mean Squared Error

PCA – Polymorphous Computing Architectures

pdf – Probability Density Function

SAT – Signature-Aided Tracking

SNR – Signal-to-Noise Ratio

CONVENTIONS

The `courier` font is used for programming code/pseudocode and also quotations (with a reduced font size).

The logarithm function base 10 is indicated simply by "log," where the natural logarithm function (logarithm base e) is indicated by "ln."

REFERENCES

[1]   William G. Coate, "Preliminary Design Review: Kinematic Tracking for the PCA Integrated Radar-Tracker Application," MIT Lincoln Laboratory Project Report PCA-IRT-4, 25 February 2003, issued 6 February 2004.

[2]   Alvin W. Drake, *Fundamentals of Applied Probability Theory*. McGraw-Hill Series in Probability and Statistics. © 1967, McGraw-Hill, Inc.

[3]   James M. Lebak, "Preliminary Design Review: PCA Integrated Radar-Tracker Application," MIT Lincoln Laboratory Project Report PCA-IRT-1, 9 April 2002, issued 6 February 2004.

[4]   James M. Lebak, Albert I. Reuther, and Edmund L. Wong; "Polymorphous Computing Architecture (PCA) Kernel-Level Benchmarks," MIT Lincoln Laboratory Project Report PCA-KERNEL-1, 23 January 2004.

REPORT DOCUMENTATION PAGE

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE		3. DATES COVERED (From - To)
27 October 2004	Project Report		

4. TITLE AND SUBTITLE

Preliminary Design Review: Feature-Aided Tracking for the PCA Integrated Radar Tracker Application

5a. CONTRACT NUMBER
F19628-00-C-0002

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

W. G. Coate and M. Arakawa

5d. PROJECT NUMBER
1084

5e. TASK NUMBER
0

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

MIT Lincoln Laboratory
244 Wood Street
Lexington, MA 02420-9108

8. PERFORMING ORGANIZATION REPORT NUMBER

PR-PCA-IRT-5

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

DARPA/ITO
3701 Fairfax Drive
Arlington, VA 22203-1714

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
ESC-TR-2004-076

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The Feature-Aided Tracker (FAT) system was developed at MIT Lincoln Laboratory as an addition to a simple kinematic tracker. The goal of FAT is to use high resolution information about a target's characteristics to aid both in the tracking and identification of targets.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT Unclassified	**b. ABSTRACT** Same as Report	**c. THIS PAGE** Same as Report	Same as Report	38	**19b. TELEPHONE NUMBER** (include area code)